23/9

9. Built a RNN

Aim: To built a RNN that learns from sequential data and predicts the next value in a sequence

Algorithm:

1) Prepare Input sequence data.
2) initialize RNN parameters and architecture
3) for each step, process input with recurrent units to capture temporal dependencies.
4) pass the final hidden state through adence layer.
5) use the trained model to predict future sequence values.

pseudo code;

input : sequence data $x = [x_1, x_2 \ldots, x_T]$
output : predicted next value Y_hat

initialize RNN parameters and weights

for each training epoch:

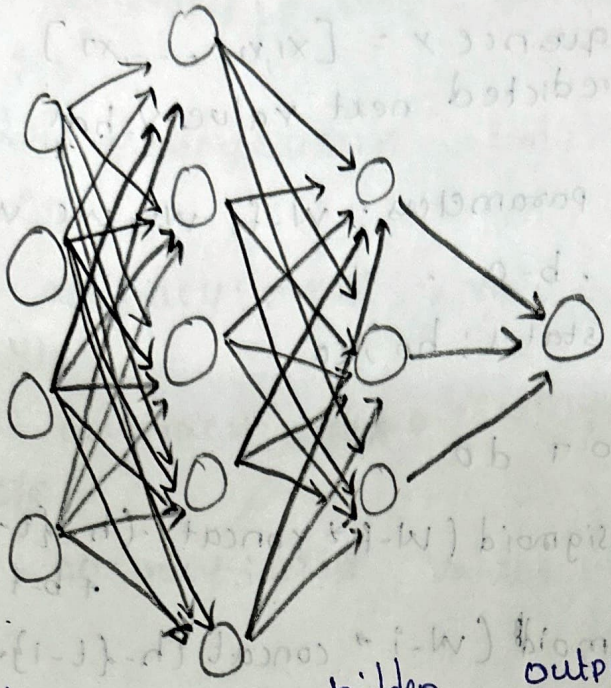
for t = 1 to T

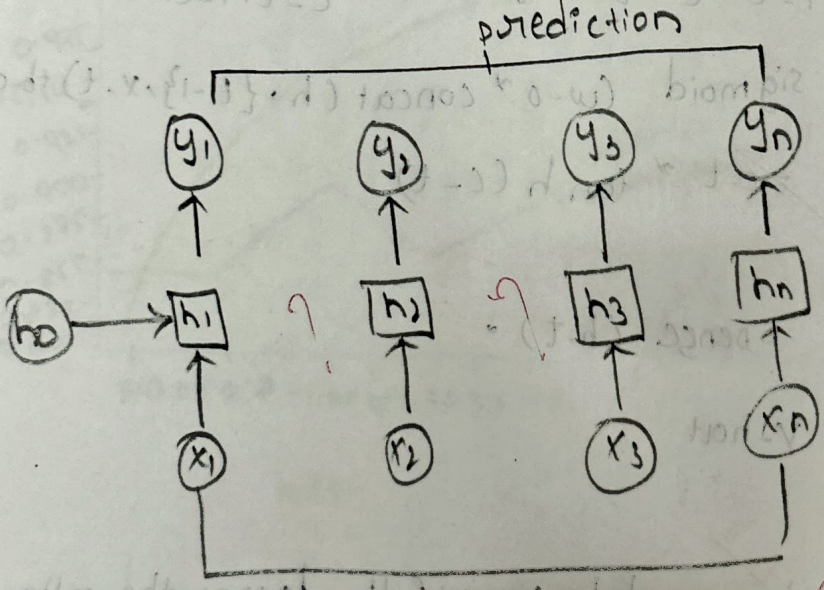
$h\_t = \text{activation}(W\_h^* h\_\{t-1\} + W\_x^* x\_t + b)$

end for

Y_hat = dense(h_T)

Recurrent neural network.





compute loss = loss (y_hat, y_true)

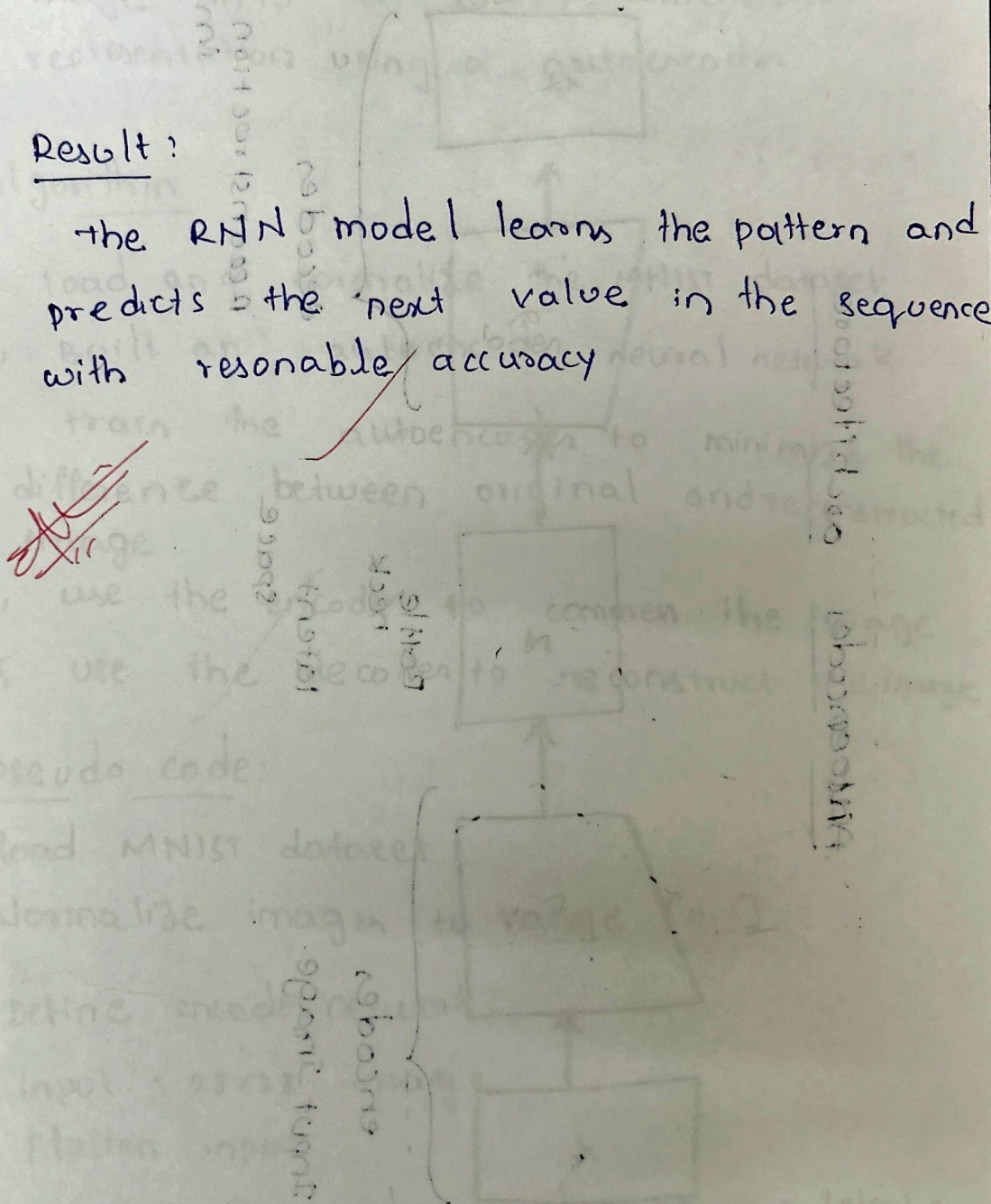
update weights using backpropagation.

predict y_hat for new input sequence

Return y_hat

Result:

the RNN model learns the pattern and predicts the next value in the sequence with resonable accuracy



Epoch1

val_acc : 0.7730 - val_loss : 0.5143

Epoch2

val_acc : 7254 - val_loss : 0.5402

Epoch3

val_acc : 0.6488 , val_loss : 0.7314

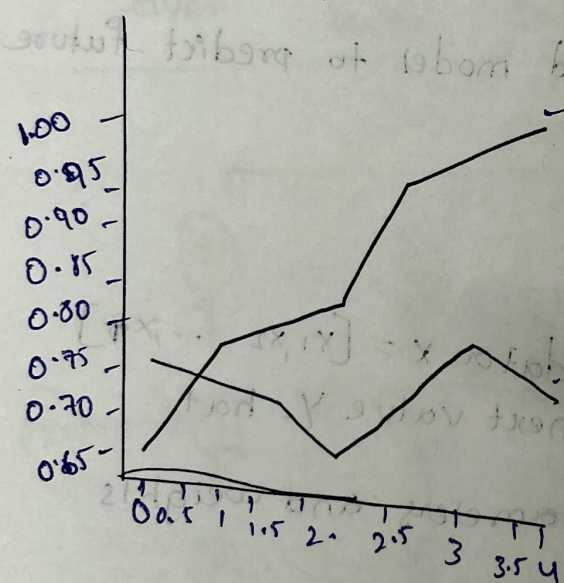
Epoch 4

val_acc : 0.7654 - val loss : 0.6499

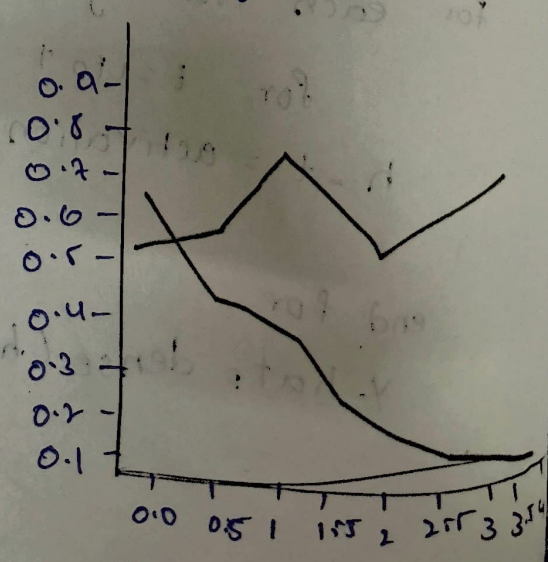
Epoch5

val_acc : 0.7350 , val loss : 0.7295

Accuracy



loss



```python
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense, Dropout
import matplotlib.pyplot as plt

# Load IMDB dataset
vocab_size = 10000
maxlen = 200

(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=vocab_size)
x_train = pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)

# Build RNN model
rnn_model = Sequential([
    Embedding(input_dim=vocab_size, output_dim=128, input_length=maxlen),
    SimpleRNN(128, dropout=0.2),
    Dense(1, activation='sigmoid')
])

rnn_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train model
rnn_history = rnn_model.fit(
    x_train, y_train,
    epochs=5,
    batch_size=64,
    validation_split=0.2
)

# Evaluate model
rnn_score = rnn_model.evaluate(x_test, y_test)
print(f"RNN Test Accuracy: {rnn_score[1]*100:.2f}% | Loss: {rnn_score[0]:.4f}")

# Plot Accuracy and Loss
plt.figure(figsize=(12,5))

# Accuracy
plt.subplot(1,2,1)
plt.plot(rnn_history.history['accuracy'], label='Train Accuracy')
plt.plot(rnn_history.history['val_accuracy'], label='Validation Accuracy')
plt.title('RNN Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Loss
plt.subplot(1,2,2)
plt.plot(rnn_history.history['loss'], label='Train Loss')
plt.plot(rnn_history.history['val_loss'], label='Validation Loss')
plt.title('RNN Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

```
Epoch 1/5
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/embedding.py:97: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
313/313 ━━━━━━━━━━━━━━━━━━━━ 33s 99ms/step - accuracy: 0.5691 - loss: 0.6649 - val_accuracy: 0.8236 - val_loss: 0.4099
Epoch 2/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 30s 96ms/step - accuracy: 0.8320 - loss: 0.3859 - val_accuracy: 0.8028 - val_loss: 0.4568
Epoch 3/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 42s 100ms/step - accuracy: 0.8403 - loss: 0.3642 - val_accuracy: 0.6352 - val_loss: 0.6383
Epoch 4/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 28s 90ms/step - accuracy: 0.7392 - loss: 0.5386 - val_accuracy: 0.5704 - val_loss: 0.6646
Epoch 5/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 28s 90ms/step - accuracy: 0.7226 - loss: 0.5610 - val_accuracy: 0.7380 - val_loss: 0.5574
782/782 ━━━━━━━━━━━━━━━━━━━━ 11s 14ms/step - accuracy: 0.7326 - loss: 0.5505
RNN Test Accuracy: 73.29% | Loss: 0.5492
```